7-15-1991

Generating just temperament with ideal rate multiplication

Andrew C. Moore

GENERATING JUST TEMPERAMENT WITH IDEAL RATE MULTIPLICATION

by

Andrew C. Moore

A thesis submitted

in

Partial Fulfillment

of the

Requirements for the Degree of

MASTER OF SCIENCE

in

Electrical Engineering

Approved by:

Prof. Names Illegible
(Thesis Advisor)

Prof. _____

Prof. _____

Prof. _____
(Department Head)

DEPARTMENT OF ELECTRICAL ENGINEERING

COLLEGE OF ENGINEERING

ROCHESTER INSTITUTE OF TECHNOLOGY

ROCHESTER, NEW YORK

July 15, 1991

Title of thesis GENERATING JUST TEMPERAMENT WITH IDEAL RATE MULTIPLICATION

I, Andrew C. Moore hereby grant permission to the Wallace Memorial Library of RIT to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Date JULY 18, 1991.

ABSTRACT

I have developed a new rate multiplication method. I call it ideal because the quality of the frequency approximation that it makes is maximized, and it extends the range of its frequency scaling factor to all the rational numbers between zero and one. In addition, each input control code produces a unique output frequency. The input code is necessarily not straight binary, so the circuit is not pin compatible with other rate multipliers.

I designed the ideal rate multiplier as a method for easily producing just tempered musical intervals. Even a three bit ideal rate multiplier produces many recognizable musical intervals.

I built a prototype of a seven bit ideal rate multiplier which plays eleven notes of the twelve tone scale. The design itself is academically interesting, and deals with recursion and rational numbers.

TABLE OF CONTENTS

■ LIST OF TABLES

LIST OF FIGURES

1. IDEAL RATE MULTIPLICATION

1.1 INTRODUCTION

Just temperament and rate multiplication are dinosaurs in their respective fields.

-

At the birth of musical harmony it was discovered that frequencies that were integer multiples of each other were pleasing to the ear when sounded together, especially if the frequency ratios only involved low primes such as 2, 3, and 5. Books such as [10] detail the mathematics of harmony.

As instruments became more sophisticated, a twelve-tone scale emerged that was based on these low primes. It appeared that all twelve intervals were roughly equal, but upon scrutiny, it was found that they were not. To avoid the trouble of retuning an instrument slightly for different keys, the "equally tempered scale" (in which all twelve intervals were the same) was proposed and adopted. Some protest accompanied this transition, but it was hard to argue with a commercial success. Instruments are now equally out of tune in all keys.

The just/equal temperament debate continues, but quietly. Research done by Linda Roberts and Max V. Mathews [9] [7] seems to indicate that while some listeners grow accustomed to the just tempered scale, others never do. It

also seems to indicate that people "have an ear" for harmonies involving higher primes than five. Specifically, their research found that subjects produced "intonation sensitivity" patterns for intervals involving a highest prime factor of seven that were similar in shape to the patterns for intervals involving a highest prime factor of five.

-

At the birth of computing, the binary rate multiplier emerged, and was proposed [8] as an economical way to do calculations without a central processing unit.

Rate multipliers "multiply" by deleting pulses in their input rate. Their output is a time-quantized approximation to an ideal time-continuous rate. The pulses that appear at the output are not evenly spaced. This uneven spacing is called "jitter".

They are called multipliers because their average output frequency is the product of the master clock frequency and the frequency scaling factor of the input code, a value between zero and one.

Use of the binary rate multiplier as a computation device is fundamentally wrong. Although it is physically able to do the job, there are few, if any, applications where some other device wouldn't be a better choice. The

binary rate multiplier did not take hold in the TTL marketplace, and was discontinued.

The rate multiplier offers a way of producing a digital approximation to a frequency. A binary rate multiplier does not do this task as well as it can be done, however. First of all, the quality of the approximation is not maximized in a binary rate multiplier. Secondly, the range of frequencies is limited. For example, the 7497 binary rate multiplier contains a six bit counter and produces from zero to sixty three pulses at its output for every sixty four (2^6) pulses input to it. The ratio of output to input frequency is n/64. The denominator is a pure power of two, so it has no odd prime factors. This is true for any binary rate multiplier.

For the purpose of frequency synthesis, the binary rate multiplier has also fallen by the wayside. Two authoritative books on frequency synthesis ,[6] and [3], lack any reference to rate multiplication.

-

I designed the ideal rate multiplier as a method for producing a digital approximation of any frequency rationally related to the top, or master, frequency that the device is run at. Every frequency producible by the ideal rate multiplier has one and only one code that will produce it. The quality of the ideal rate multiplier's approximation is also as good as possible.

As a general purpose frequency synthesizer, the ideal rate multiplier has deficiencies that binary rate multipliers do not have. One deficiency is the input coding scheme, which is not a common coding scheme like straight binary. Another more serious deficiency is that it does not have an even distribution of output frequencies. Although it is able to produce any frequency, many stages are required for it to get fine frequency resolution throughout the whole range.

The ideal rate multiplier readily produces just tempered musical intervals. I designed it with this application in mind. Given that just temperament and rate multiplication are both commercial failures, it is easy to question developing a device that marries the two. Why did I do it? For me, it was a mountain to climb. I did it because it was there. I did it to explore the link between mathematics and music, a link that I feel is very real, but still generally misunderstood.

1.2 HISTORICAL REVIEW OF THE RATE MULTIPLIER

The 7497 is a six bit binary rate multiplier (or BRM). Most modern logic families (LS, HCT, ACT, FAST) do not include the 7497 in their lineup. According to [11] and [12], it is only available from Texas Instruments in the obsolete standard TTL family. It never even graduated to the low-power Shottky (LS) family, which is also considered obsolete by some.

Originally, as in [8] and [5], the BRM was marketed as a way of performing calculator-like operations: multiplying, dividing, taking square roots, exponentials, sines, cosines, and other functions. More direct algorithms exist for all of these tasks, though, and the BRM is now a poor choice for these applications.

Don Lancaster, in [5], admitted that the BRM was "somewhat slow", but said that "they do not seem to be used much because designers are not familiar with them..." He maintained, however, that "the rate multiplier offers an interesting approach to a wide variety of practical problems."

I agree that BRM's are interesting devices, and perhaps they are still unused because designers are still unfamiliar with them, but it seems far more probable that BRM's are no longer a cost effective solution for computation.

Texas Instruments' application notes, [8], give a clue to the extinction of the BRM. They start out "Some processing does not need the power of a c.p.u. or a complete microprocessor.." The cost of a c.p.u. has dropped drastically since then. It may be that cheap processors have closed these devices right out of their market.

The BRM is a poor computation device because it is slow, and the low entropy of a BRM's output is what makes it this way. This deficiency of the BRM as a computing device merits further explanation.

A BRM is basically a synchronous digital-to-frequency converter. Its output is a square wave that is temporally quantized. Temporal quantization results in noise which constrains the information-carrying ability of the BRM's output.

If an analog square wave frequency generator is directed to output a different frequency, the change in frequency is completely measurable after only one cycle. The information carrying capacity of an analog frequency generator is therefore limited only by the channel noise. In contrast, a BRM makes its own noise. If a BRM is directed to output a different frequency, its output must be monitored until its internal counter cycles through all of its values to completely measure its output frequency. A BRM followed by a frequency-to-voltage converter can be used as a crude

digital-to-analog converter, but a great degree of low-pass filtering is required to get an output voltage that is stable to one least significant bit. The time constant of the low-pass filter is at least on the order of 2^n cycles for an n-bit circuit. This means that each additional bit of information conveyed as frequency costs an octave of bandwidth, given the BRM is running at its maximum rate. (It usually is to get maximum throughput.)

In general, more precision takes more time. Some parallel math processors are an exception to this rule though, and can be extended in precision without loss of throughput simply by "widening" them. Many hardware and software math algorithms require one gate delay or one iteration per bit, so their computation time often varies proportionally to the number of bits of precision.

BRM circuits are worse than most in this regard. They slow down exponentially as more precision is required. Since they often require at least one cycle through an n-bit counter to complete, their computation time often varies with 2^n. Sometimes, as in [8], BRM circuits require 2^{2n} cycles to complete. Twelve bit precision requires thousands of cycles. Twenty four bit precision requires millions of cycles. Attaining great (such as 64 bit) precision with a BRM circuit would practically take forever.

1.2.1 FREQUENCY GENERATION APPLICATIONS

Due to its poor performance as a computation device, frequency generation is probably the only area where a rate multiplier would be considered useful nowadays. Using the BRM as a frequency generator is straightforward, and some designers used it just for frequency synthesis.

For example, Dr. Richard Campbell, [2] a professor at the State University of New York at Buffalo's department of Communicative Disorders and Sciences (now retired), designed and built a just-tempered organ using BRMs. For this application, he simply required programmable frequency synthesis.

Dr. Campbell's organ was able to play a just tempered scale in any key, and could also play in equal temperament. A diode matrix connected a key selection bank of pushbuttons to the BRMs' data inputs. He said that the audible difference between equal and just temperament was extremely noticeable, with just temperament being much more pleasant.

Marketing the organ, on the other hand, appeared to be impossible. In general, the consumer did not appear to care for just temperament any more than for equal temperament.

As far as the jitter in the outputs was concerned, it was audible, but Dr. Campbell said that a certain amount of jitter was desirable because the jitter added a little randomness to the sound and made it more life-like.

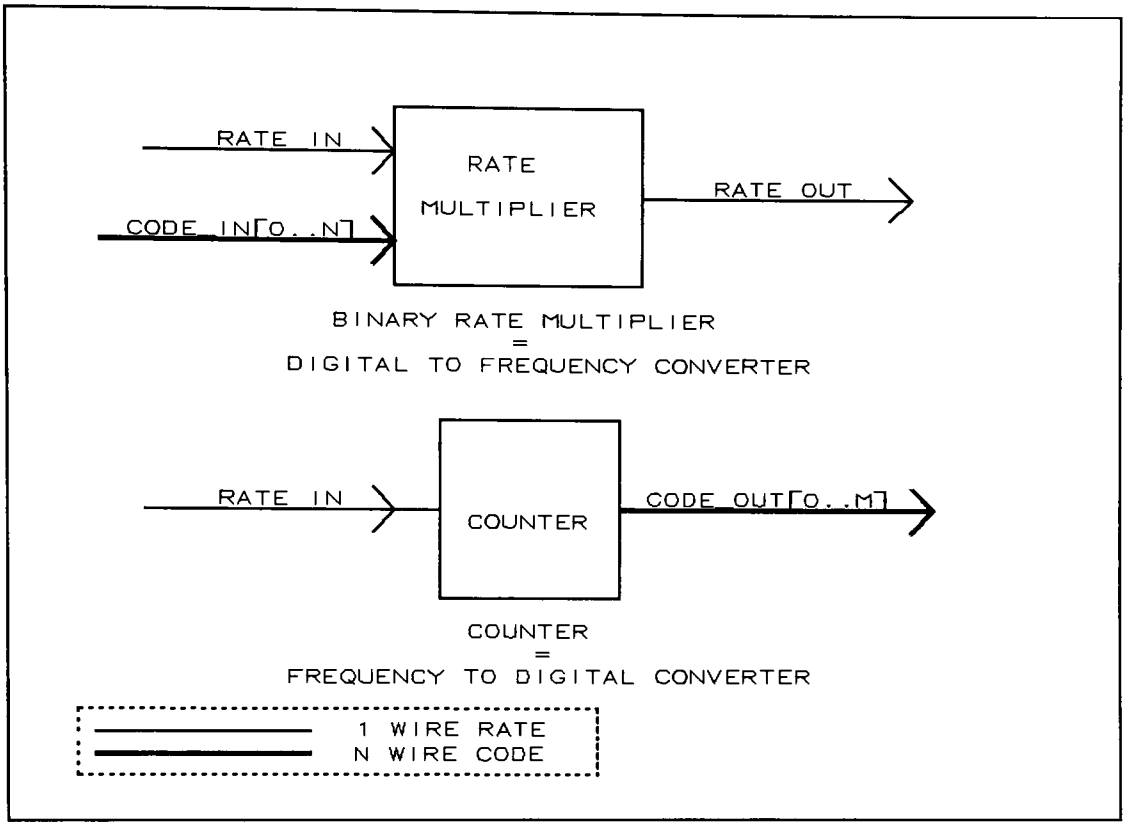

Figure 1: Binary Rate Multiplier and Counter

1.2.2 COMPUTATIONAL MULTIPLIER APPLICATIONS

For many computation purposes, the frequency generation aspect of the BRM is simply an annoyance. Ideally, the BRM operates at infinite frequency in these cases. In practice, it is usually clocked as fast as physically possible.

The inputs and outputs of a computational BRM circuit are generally binary code, not frequency. Computational BRM applications typically deal with two types of signals, unsigned integer and frequency.

Counters play a crucial role in computational BRM circuits. The inputs to computational BRM circuits generally go to BRMs directly, or are preloaded into counters. The outputs are usually counter values.

BRMs can convert a binary integer to a frequency, or a frequency to another frequency. Making a feedback circuit with a BRM in the loop, however, requires a device that converts a frequency to an integer, and counters do this. A counter acts as an integrator if free-running or a low-pass filter if it is reset, run, and latched. Up-down counters can take the difference of two frequencies.

Figure 1 illustrates the BRM and counter. The BRM and binary counter, as a pair, perform bidirectional data conversion between two types of information, binary code and frequency. They can also add, subtract, multiply, and integrate. The counter and BRM are basic building blocks for modeling analog computer equations.

Many BRM applications involve a state machine whose state at some time is of interest. Typically, they involve converting an input code to a frequency and injecting this frequency into the loop of a feedback circuit. This is accompanied by waiting for the value of the output, usually a counter value, to attain significant precision.

BRMs can be used so that the path that the circuit takes from some initial condition simulates a differential

equation. The input variable is the amount of time that the circuit is allowed to run. These applications are called mathematical functions in [8], and can compute simple and complex exponentials and other trigonometric functions.

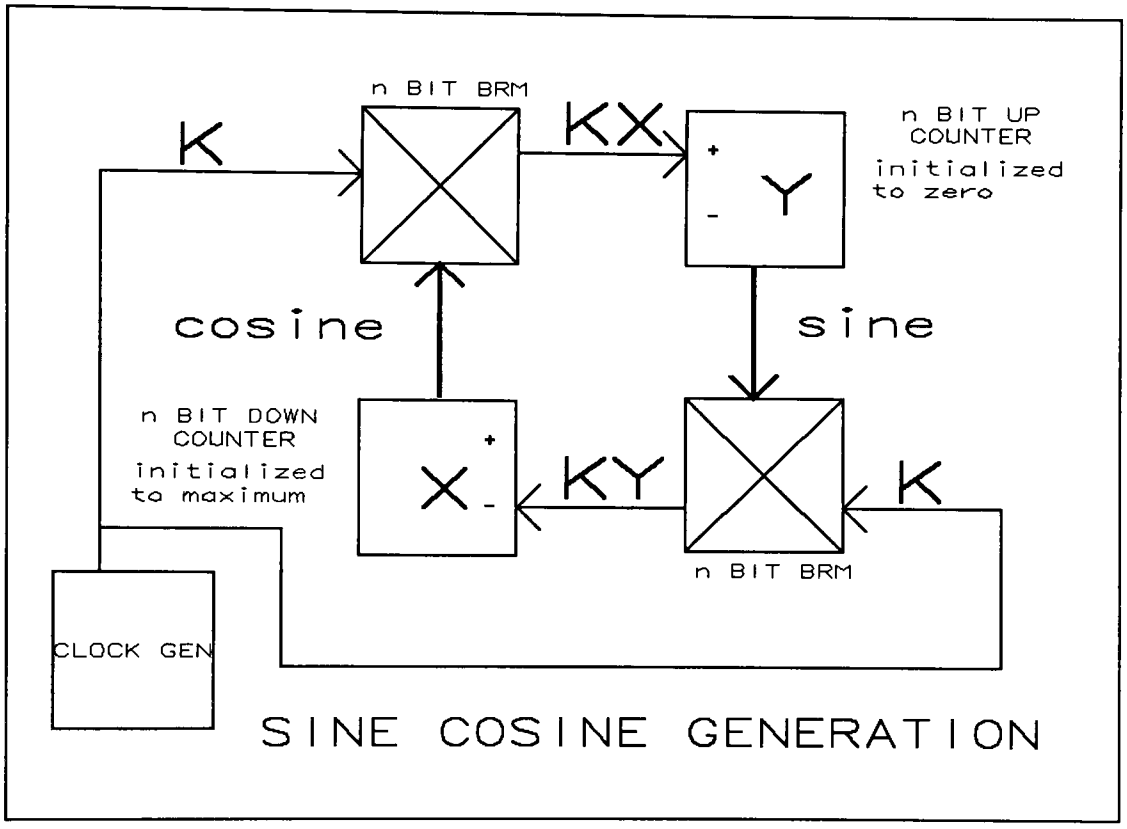

Figure 2: Sine and cosine generation with a binary rate multiplier. (from [8])

Figure 2, taken from [8], illustrates sine and cosine generation. This circuit models the differential equation pair:

$$dx/dt = -ky,$$

$$dy/dt = kx$$

The x counter is initialized at its maximum, and generates a cosine shape. The y counter is initialized at zero and does the sine. This circuit does the first quadrant only.

BRM analog computer circuits are sometimes designed to stabilize to a certain value as time goes to infinity. If the desired output is this final state, [8] calls it an arithmetic operation. They also give examples of circuits that can be used to divide, do rational roots, and find other arithmetic functions of the control inputs.

Figure 3, taken from [8], shows how to generate the square root of a number with a BRM circuit. Two separate clock phases are needed, so that the counter does not receive simultaneous up and down pulses.

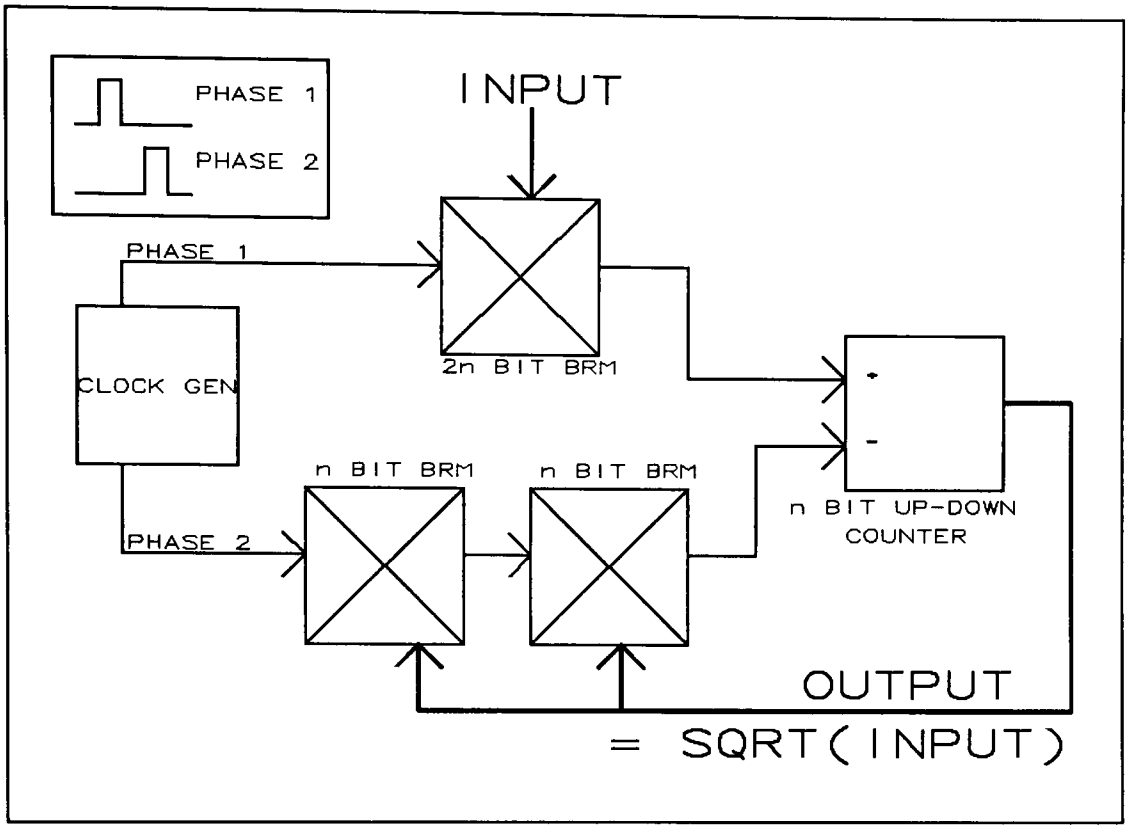

Figure 3: Square root generation with a binary rate multiplier. (from [8])

In the square root circuit, the counter can be thought of as an integrator, or an op amp. The op-amp analogy is valid because the circuit ideally runs at infinite frequency and stabilizes instantly. Op-amps and integrators have infinite DC gain, so, upon stabilizing, the inputs are both equal.

Over time, the circuit in Figure 3 models the differential equation

$$dx/dt = kI - k(x^2).$$

The solution of this equation is not what is of interest, though. Since x is positive, the feedback is negative, and causes x, the counter value, to stabilize. When dx/dt reaches zero, the equation reduces to

$$I = x^2, \text{ or}$$

$$x = \text{sqrt}(I).$$

The counter stabilizes at the square root of the input value.

Figure 4, taken from [1], shows the analog version of this circuit in figure 3. It is much simpler. Two of the BRMs in figure 3 are unnecessary in the analog version.

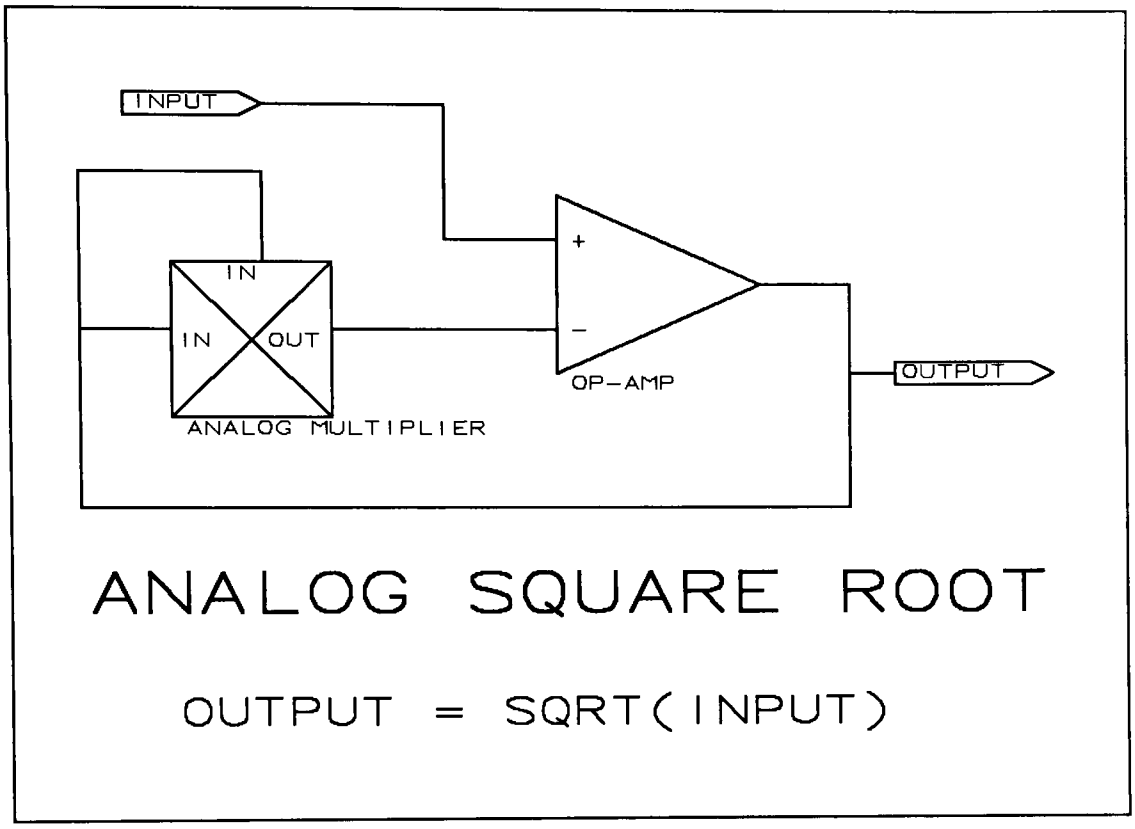

Figure 4: Square root generation with an analog multiplier. (from [1])

The analog version needs protection from latch up. Should the op amp output go negative, the multiplier output will go positive, and negative feedback will be lost.

1.3 THEORY OF THE IDEAL RATE MULTIPLIER

My ideal rate multiplier generates any ratio of pulses to missing pulses, and spaces its output pulses as evenly as possible. The core of my ideal rate multiplier is a circuit that produces a string of p 1's and q 0's, p and q relatively prime, or both unity, and the 1's and the 0's spaced as evenly as possible. For p pulses and q missing pulses, the ratio of pulses to missing pulses lies between zero and infinity. The net frequency produced, however, is p pulses out of p+q total "pulse opportunities", and lies between zero and one.

Before I could design a circuit that does this, I needed an algorithm that spaced p 1's and q 0's "as evenly as possible".

1.3.1 THE ALGORITHM

The problem is to arrange p X's and q 0's, p and q relatively prime, in a ring so that they are spaced as evenly as possible.

The solution is explained as two algorithms. First, one that is simpler conceptually (easier to understand) and then one that is simpler physically (easier to build). Both algorithms work roughly the same, though.

The end goal is construction of a pattern based on a certain p and q. To determine the pattern, we take the p, q pair and recursively reduce it (to a new p and q). The first algorithm uses type I reductions, and the second algorithm uses type II reductions. Eventually, a reduction results in a trivial case, for which a pattern is known. Then a reconstruction starts with this known pattern. All of the reductions are applied in reverse. This is expansion. The first algorithm uses type I expansions and the second algorithm uses type II expansions. Complete expansion yields the desired pattern.

1.3.1.1 THE FIRST ALGORITHM

If either p or q is 1, the case is trivial (See figure 5). Any arrangement will do. Non-trivial cases have p>1 and q>1.

Any non-trivial case can be reduced to a similar problem of lesser order by using the following logic:

First of all, either p is less than q or q is less than p. The one that is smaller is the minority. To be evenly spaced, minorities can only occur by themselves. In this respect, minorities can be considered to be "spacers." This simplifies the problem to putting some correct number of majority tokens in between these lone minorities. In the discussion that follows, the one or more majorities that occur between minorities are called "subgroups."

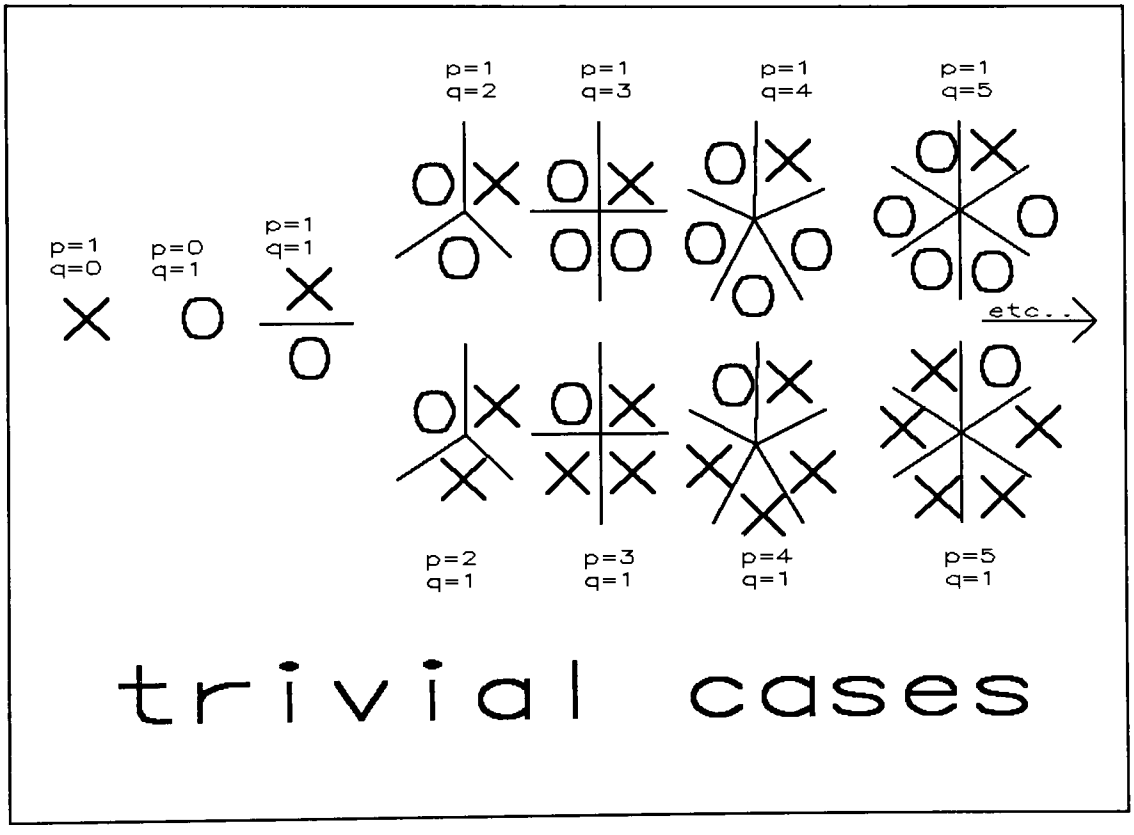

Figure 5: Trivial cases of ideal spacing.

A second observation is that for any given stage of reduction, a subgroup's size can only be one of two possible values which differ by one. Dividing the number of majorities by the number of minorities yields the average subgroup size. The quotient is the size of the smaller. The remainder of this division is the number of subgroups with an extra majority.

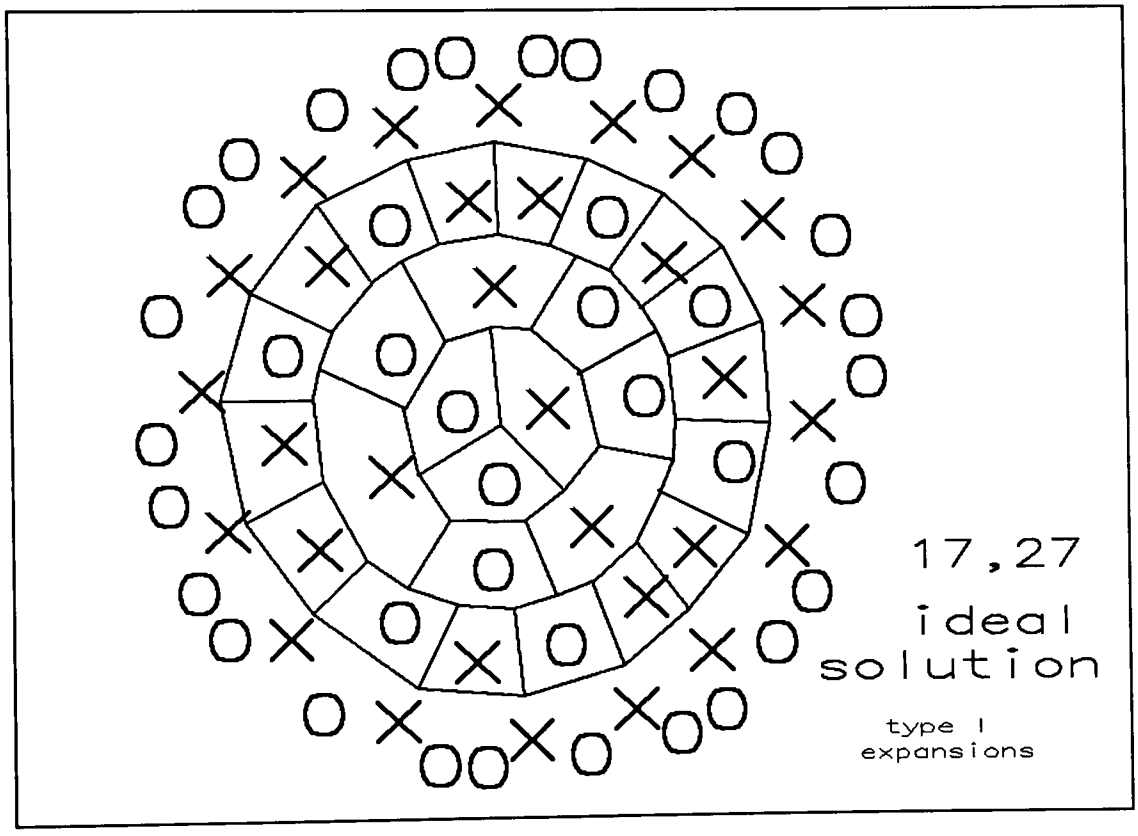

Figure 6: A complex example of an ideal solution using type I expansions.

Now the problem is reduced to spacing these two types of subgroups as evenly as possible, which is a problem of the same type as the original one, but of lesser order. This reduction technique is repeatedly applied until a reduction results in a trivial case. This completely specifies the solution.

1.3.1.2 AN EXAMPLE USING THE FIRST ALGORITHM

A good example is p=17, q=27.

p=17, q=27 has 17 subgroups, 10 large and 7 small.

p=10 q=7 has 7 subgroups, 3 large and 4 small.

p=3, q=4 has 3 subgroups, 1 large and 2 small.

p=1, q=2 is trivial.

At this point, expansion starts. Each reduction is applied in reverse and the pattern is constructed.

Figure 6 shows the expansion in a multi-ring format that I have devised. Separating lines in a ring expand into the minorities of the next ring out. X's expand to large subgroups, and O's correspond to small subgroups of the next ring out.

1.3.1.3 PHYSICAL NEED FOR A SECOND ALGORITHM

Two pieces of information are extracted from each reduction in the first algorithm. Each reduction can be represented by the type of the minority and the size of a small subgroup. The minority type is one of two

possibilities, so it is one bit of information. The small subgroup may be any size, though. The previous example, p=27, q=17, is a hand-picked example because there are always one or two majorities in subgroups at every level. In the general case this is not always true, and there are an infinite number of possibilities in each reduction. This makes it hard to directly implement the first algorithm in a physical circuit. If each reduction is implemented by a single stage, a possibly infinite amount of information can enter each stage.

I looked for a physical solution that took the form of a counter, with one data bit entering each stage, one state bit per stage, and control lines rippling between stages where necessary. Since the algorithm is recursive, each stage could be identical and perform one more reduction in the solution. One data and one state bit per stage would be the simplest architecture.

To conform the algorithm to this physical structure, I needed to model the solution such that each reduction had only two possibilities. I modified the first algorithm to a second one where information is extracted only one bit at a time.

1.3.1.4 THE SECOND ALGORITHM

The second algorithm is exactly like the first algorithm, except for major differences. The single

minorities, which are spacers in the first algorithm, are the large subgroups in the second algorithm. Small subgroups, to be smaller than large subgroups, must therefore be null. Majorities, which make up the subgroups in the first algorithm, are considered the "spacers" in the second algorithm.

To go "out one ring" in the second algorithm, place one majority at each "space" in the ring. Replace each 0 with a small subgroup (which is zero length here), and replace each X with a large subgroup. Figure 7 shows how the second algorithm works.

The only trivial cases for the second algorithm are an all zeros case and an alternating pattern. The alternating pattern is the pattern that construction always starts with, and I call it the "top bit pattern."

One reduction in the first algorithm will match two or more reductions by the second algorithm. The number of second algorithm reductions required is equal to the size of a small subgroup in the first algorithm, plus one. For example, if a ring in the first algorithm has a small subgroup size of ten, then it will take eleven second algorithm reductions to reduce it to the pattern that one first algorithm reduction would reduce it to.

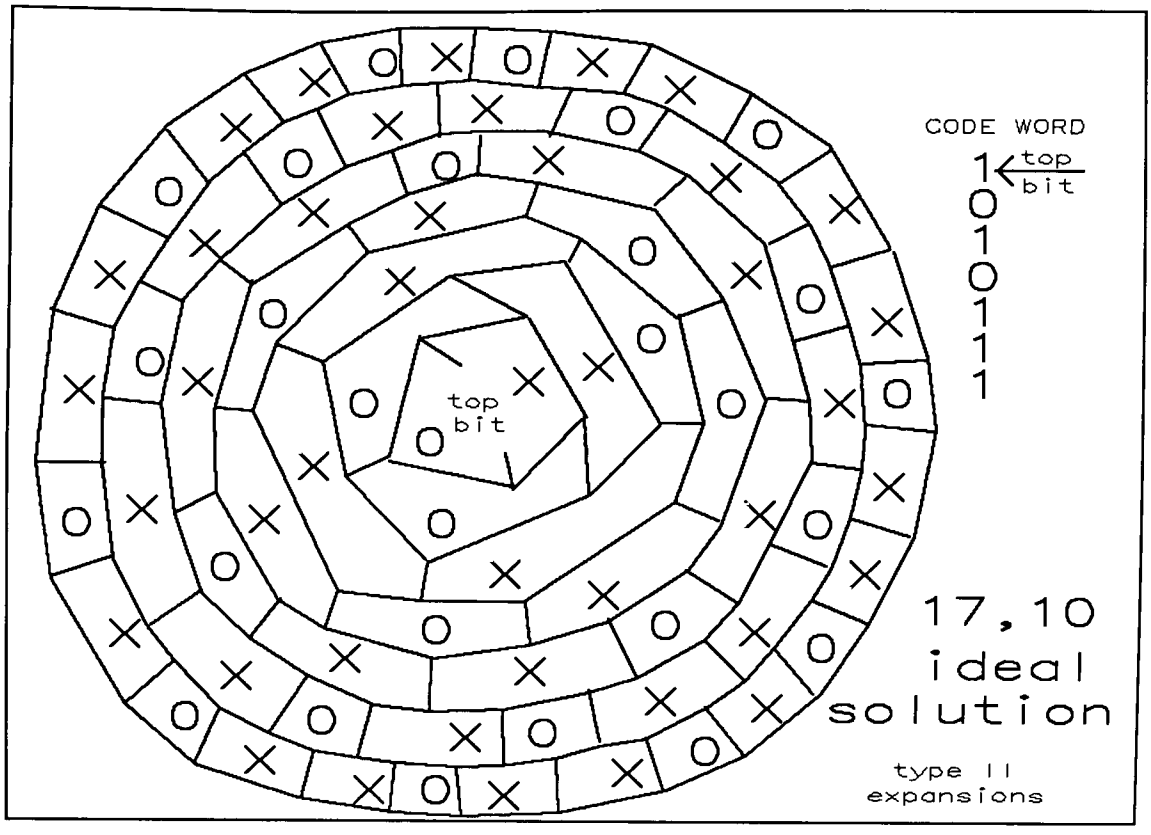

Figure 7: A complex example of an ideal solution using type II expansions.

1.3.1.5 AN EXAMPLE USING THE SECOND ALGORITHM

Here is how 27,17 gets reduced by the second algorithm:

27,17 has 17 large subgroups and 10 small subgroups.

17,10 has 10 large subgroups and 7 small subgroups.

These two reductions complete the first reduction of the original algorithm.

10,7 has 7 large subgroups and 3 small subgroups.

7 ,3 has 3 large subgroups and 4 small subgroups.

These two reductions complete the second reduction of the original algorithm.

4 ,3 has 3 large subgroups and 1 small subgroup.

3 ,1 has 1 large subgroup and 2 small subgroups.

3 ,1 was originally considered a trivial case.

These two reductions complete the third reduction of the original algorithm. The previous reduction and the remaining reduction reflect the difference between what is considered a trivial case in each algorithm.

2 ,1 was also originally considered a trivial case, but requires another reduction to become the top bit pattern.

2 ,1 has 1 large subgroup and 1 small subgroup.

1 ,1 is trivial (the top bit pattern).

At this point, expansion begins. Figure 7 shows the expansion, except for the very last ring.

1.3.2 THE CIRCUIT

My ideal rate multiplier consists of two parts. The first part is the multiplier core, which consists of stages chained together. The core implements the equal spacing algorithm. The second part is the frequency generation circuitry, which produces the frequency output based on the output of the multiplier core.

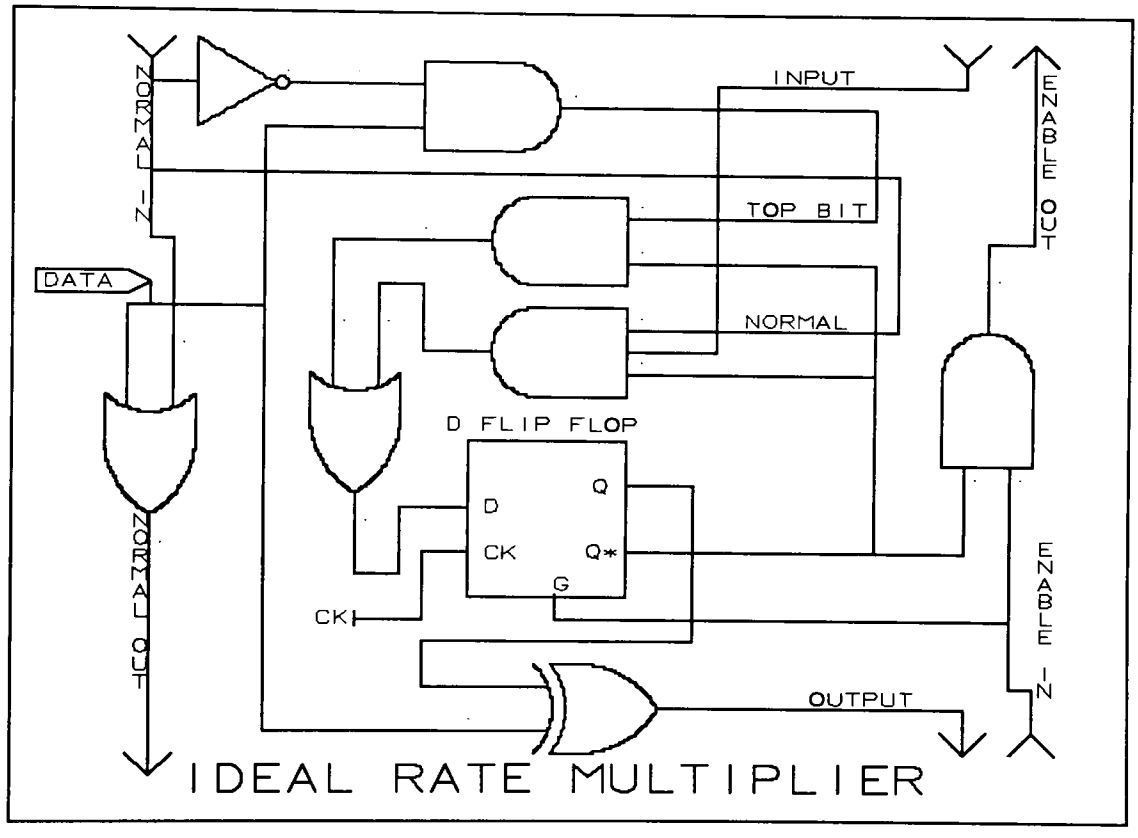

Figure 8: One stage of an ideal rate multiplier.

During explanation of the algorithm, I used "X's" and "0's". In the description of circuit that implements it, I

call the X's "1's" instead to be consistent with conventional logic.

1.3.2.1 THE MULTIPLIER CORE

Figure 8 shows one stage of the ideal rate multiplier, and figure 9 shows how several of these stages connect together. The rightmost line, enable, travels up. This causes upper stages in the counter to be clocked less frequently than lower stages.

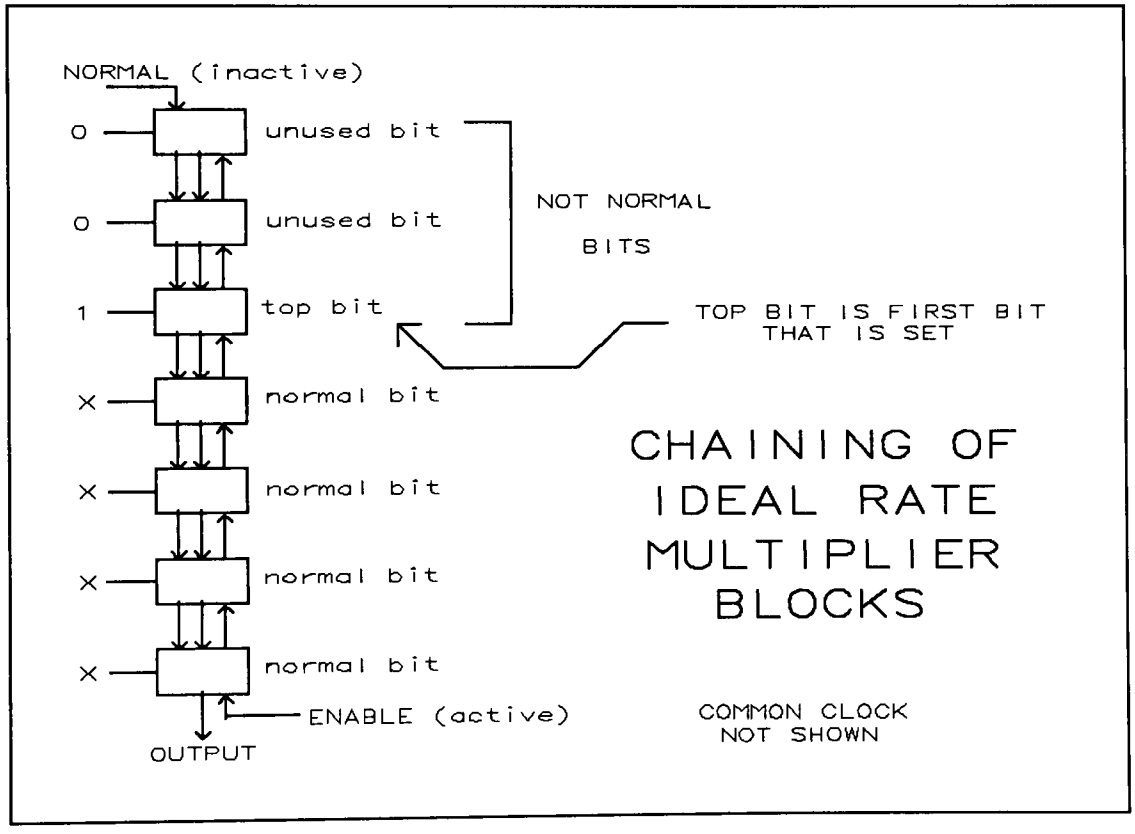

Figure 9: Cascading ideal rate multiplier stages.

I call the leftmost control line the "normal" line. The purpose of the normal line is to identify the "top bit" of the ideal rate multiplier. In effect, it is a ripple blanking line. It starts in an inactive state at the top of the multiplier, and is passed down from stage to stage until it reaches the bottom stage. If the normal line entering a stage is inactive and the data bit entering the stage is also inactive, then that stage is disabled and passes an inactive normal line to the stage below it. Stage operation is disabled in this manner until the first set data bit is encountered.

The first stage encountered along the normal line that has a set data bit is the "top stage" of the rate multiplier, and the data bit entering it is the "top bit". This stage acts differently from any other stage in the counter. The top stage simply toggles at the clock if it is enabled, independently of the output of the stage above it. All stages below the top stage are normal counter stages.

When enabled, the state bit of a normal stage clocks to a new value based purely on itself and the output of the stage above it. If it is clear and the output above is clear, it will stay clear. If it is clear and the output of the stage above it is active, it will set itself. If it is set, it will clear when next enabled. Since a clear bit may

or may not set, but a set bit will always clear, the state bit will be clear a majority of the time.

The state bit of each normal stage determines if the output is a majority or a minority. A 0 in the state bit means that the output is a majority, and a 1 means that the output is a minority. When the state bit is set, the stage blocks the enable to the stages above it. It does this because at this point a minority is inserted in the output, so an extra clock cycle is needed.

The data bit that enters each stage selects the type of majority in each stage's output. If the data bit is 0, the output of that stage is a majority of 0's. Otherwise, the output is mostly 1's. This output is passed on to the next stage.

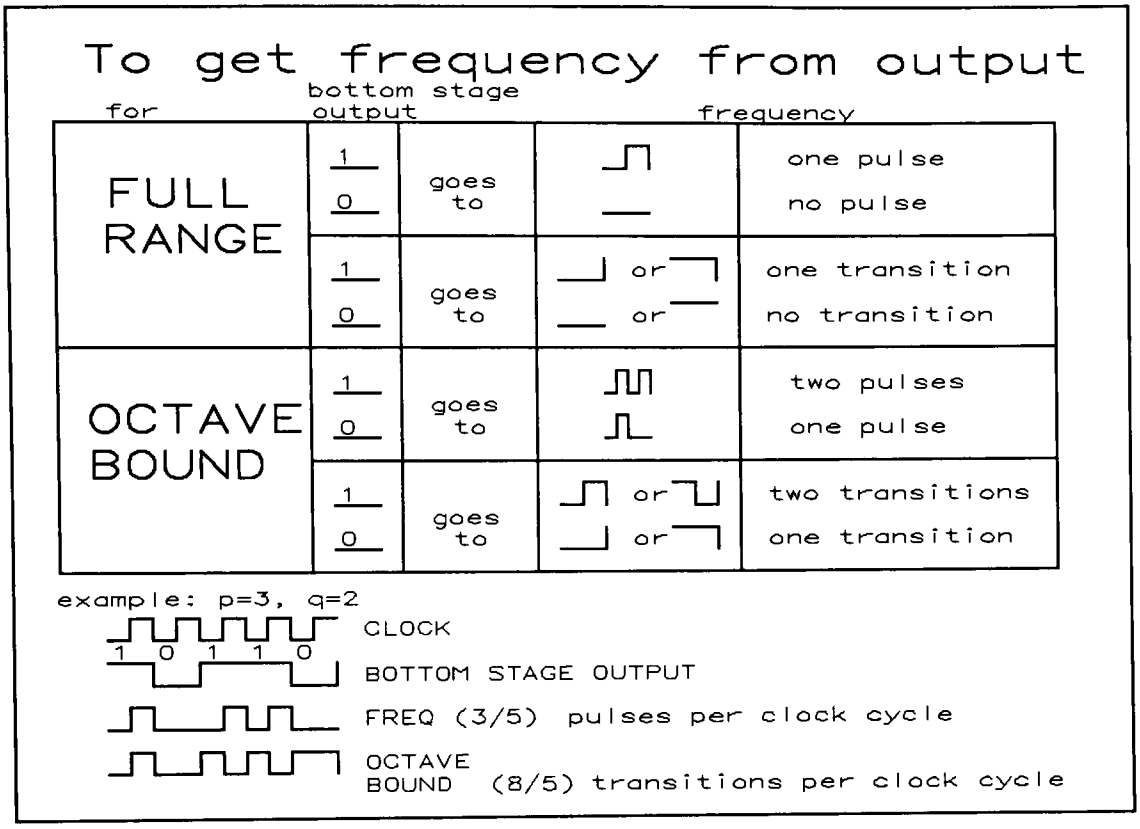

Figure 10: Deriving full range and octave bound frequency from the bottom stage output

1.3.2.2 FREQUENCY GENERATION CIRCUITRY

The output of the bottom stage can get converted to a frequency in several possible ways. Full range multipliers convert 1's to single, and 0's to missing, pulses or transitions. Octave-bound multipliers convert 1's to pairs of, and 0's to single, pulses or transitions. Figure 10 shows sample output waveforms for all four of these methods. Figure 11 shows circuitry for transitional frequency output.

Both full-range and octave bound output may be obtained from this circuit.

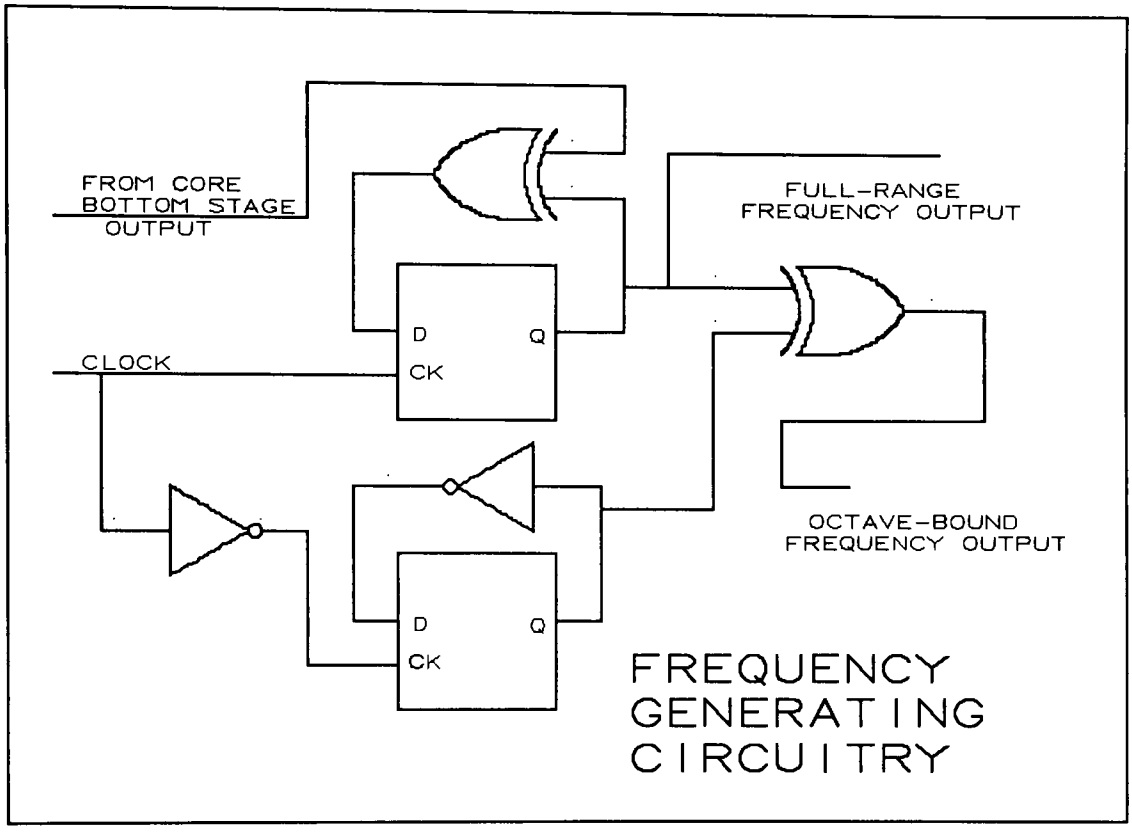

Figure 11: Frequency generation circuitry

1.3.2.3 INPUT / OUTPUT RELATIONSHIP

The simplest patterns that can be produced are shown in figure 12. All complex patterns have simple patterns at their center, so understanding the circuit operation starts with understanding these simplest patterns.

If all data bits are off, the top bit line never gets set and the output of the bottom stage is a 0.

If only the very bottom data bit is set, the bottom stage is the "top bit", and outputs an alternating 10 pattern, also called the "top bit pattern"

If the next to the bottom data bit is set, this stage is the top bit. If both states are initially zero {00}, the top stage will output a 1. At the next clock, both stages will toggle and the new state will be {11}. Since the bottom bit is set, the enable is not passed upwards. At the next clock, the bottom bit clears and the state is {10}. The top stage outputs a 0 now, so the bottom stage stays clear and enables the top stage, which toggles back to a 0, and we are back to {00}. The output of the bottom stage is 010 if the bottom data bit is clear, and 101 if the bottom data bit is set.

If only the nth data bit is set, and the nth stage output is a 1, on the next cycle, it will clear, but the state and stage output below it will get set and block the clock enable. The next clock will clear this output but set

the next one below. This set output will ripple down until it finally hits the main output. When this bit finally ripples out of the bottom, the enable ripples up to the nth bit. The next clock sets the top bit's output high again.

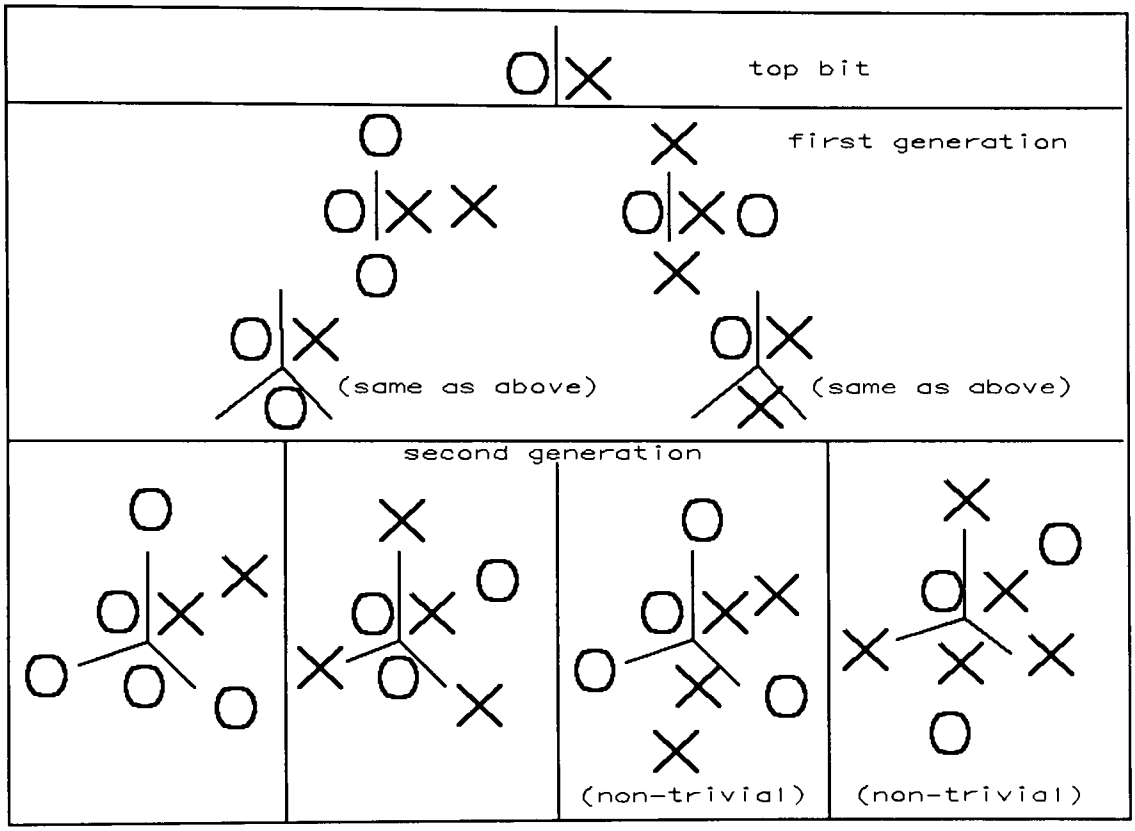

Figure 12: The first two digital generations

It is easy to derive all ideal rate multiplier output patterns and input codes. Each pattern (except for the all-zeros pattern) reduces to the "top bit" pattern. To get a new code-pattern pair from one old code-pattern pair, follow this procedure: Shift the old code word left by one bit,

shifting in a zero. This yields the first new code. Now, replace every 1 in the old pattern with a 10 pair. This yields the first new pattern.

Complementing every bit in this pattern and replacing the new zero in the code word with a 1 yields a second new code-pattern pair.

The following pages have two tables that show the output patterns that are generated by all different code words up to five bits in length. The first one is sorted by code word, and the second is sorted by the ratio of 1's to 0's, with the patterns that are a majority of 0's listed first.

Table 1: Five bit Ideal rate multiplier output patterns, listed in order of increasing code word.

input code	output pattern	1:0 ratio
00000	0	0:1
00001	10	1:1
00010	100	1:2
00011	011	2:1
00100	1000	1:3
00101	0111	3:1
00110	01010	2:3
00111	10101	3:2
01000	10000	1:4
01001	01111	4:1
01010	0101010	3:4
01011	1010101	4:3
01100	0100100	2:5
01101	1011011	5:2
01110	10010010	3:5
01111	01101101	5:3

table continues on next page...

Table 1 continued...

input code	output pattern	1:0 ratio
10000	100000	1:5
10001	011111	5:1
10010	010101010	4:5
10011	101010101	5:4
10100	0100100100	3:7
10101	1011011011	7:3
10110	10010010010	4:7
10111	01101101101	7:4
11000	010001000	2:7
11001	101110111	7:2
11010	100101001010	5:7
11011	011010110101	7:5
11100	10001000100	3:8
11101	01110111011	8:3
11110	0101001010010	5:8
11111	1010110101101	8:5

(end of table)

Table 2: Five bit ideal rate multiplier output patterns, listed in order of increasing output frequency.

input code	output pattern	1:0 ratio
00000	0	0:1
10000	100000	1:5
01000	10000	1:4
11000	010001000	2:7
00100	1000	1:3
11100	10001000100	3:8
01100	0100100	2:5
10100	0100100100	3:7
00010	100	1:2
10110	10010010010	4:7
01110	10010010	3:5
11110	0101001010010	5:8
00110	01010	2:3
11010	100101001010	5:7
01010	0101010	3:4
10010	010101010	4:5

table continues on next page ...

Table 2 continued ...

input code	output pattern	1:0 ratio
00001	10	1:1
10011	101010101	5:4
01011	1010101	4:3
11011	011010110101	7:5
00111	10101	3:2
11111	1010110101101	8:5
01111	01101101	5:3
10111	01101101101	7:4
00011	011	2:1
10101	1011011011	7:3
01101	1011011	5:2
11101	01110111011	8:3
00101	0111	3:1
11001	101110111	7:2
01001	01111	4:1
10001	011111	5:1

(end of table)

1.3.2.4 MUSICAL PROPERTIES OF THE MULTIPLIER

A three stage multiplier with octave binding circuitry has a total range of eight notes (2 to the third.)

Table 3 shows the musical values of these notes:

Table 3: Three bit ideal rate multiplier output with musical note names listed.

Input code	output pattern	1/0 ratio	1:0 +1	note	note name
0	0	0:1	1:1	C	do
100	1000	1:3	5:4	E	mi
10	100	1:2	4:3	F	fa
110	01010	2:3	7:5	F#	
1	10	1:1	3:2	G	so
111	10101	3:2	8:5	Ab	
11	011	2:1	5:3	A	la
101	0111	3:1	7:4	Bb	

Table 4 shows musical input codes and output patterns for an eight bit ideal rate multiplier. An eight bit ideal rate multiplier is able to hit every halftone in the octave except for the first.

Table 4: Eight bit ideal rate multiplier output with musical note names listed.

Input code	output pattern	1/0 ratio	ratio +1	note	note name
0	0	0:1	1:1	C	do
10000000	100000000	1:8	10:9	D	re
1000000	10000000	1:7	9:8	D	re
1000	10000	1:4	6:5	Eb	
100	1000	1:3	5:4	E	mi
10	100	1:2	4:3	F	fa
110	01010	2:3	7:5	F#	
1	10	1:1	3:2	G	so
111	10101	3:2	8:5	Ab	
11	011	2:1	5:3	A	la
101	0111	3:1	7:4	Bb	
11001	011101111	7:2	16:9	Bb	
1001	01111	4:1	9:5	Bb	
1000001	01111111	7:1	15:8	B	ti

1.4 THE PROTOTYPE 8 BIT IDEAL RATE MULTIPLIER

I built a prototype to demonstrate the musical possibilities of the ideal rate multiplier, so it includes octave binding circuitry and octave selection switches. An eight ohm speaker is the output frequency monitor. Each key combination yields a different note within a one octave band, and multiple octaves can be heard simultaneously.

The octave binding circuit in the prototype produces a pair of pulses if the output is active and a single pulse if the output is inactive. (Another way to "octave bind" the output frequency is to use a full range output circuit and keep the bottom data bit set all the time.)

1.4.1 SCHEMATICS

I constructed the prototype using programmable array logic, choosing the National 16R4 as an appropriate device. Two stages are implemented in the pal, with some of the pins buried in equations. Figure 13 shows the block diagram of the pal circuit. Figures 14, 15 and 16 show the schematics in a hierarchy. Figure 14 shows the overall view. Figure 15 shows the IRM board in slightly finer detail, and 14 shows the schematic of the core.

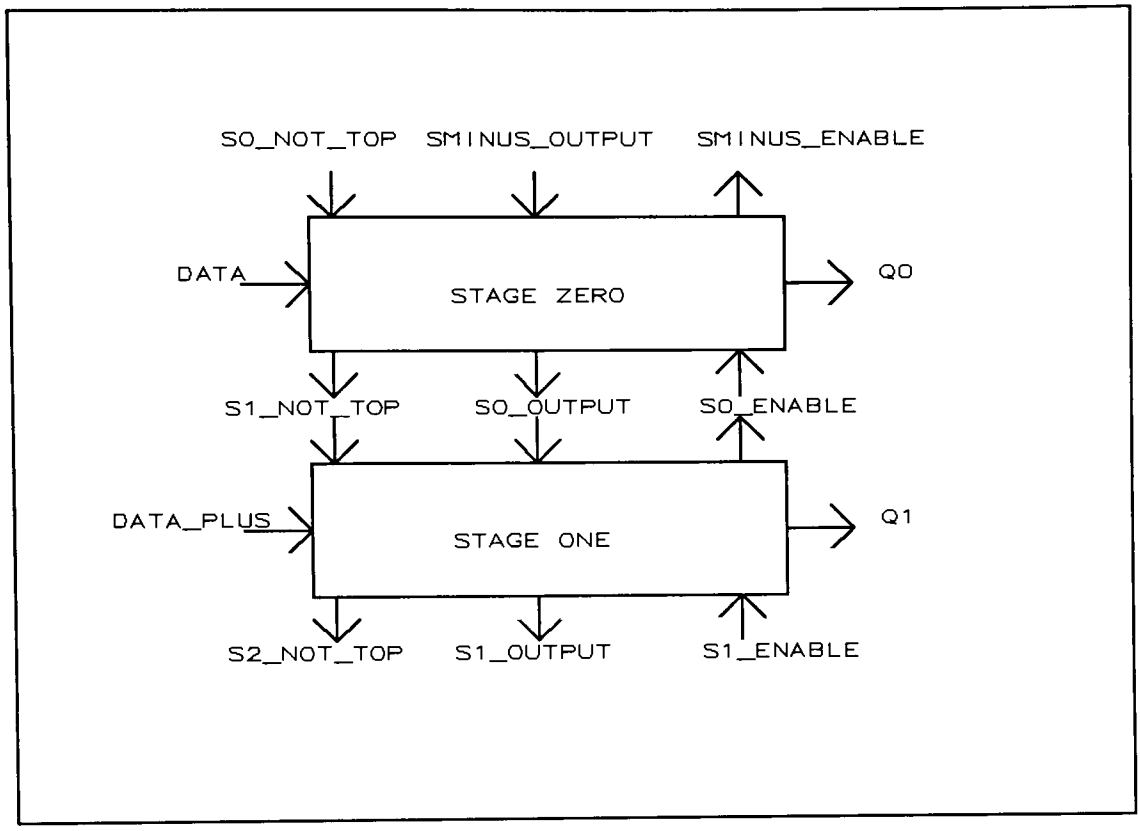

Figure 13: Block diagram of pal used in prototype.

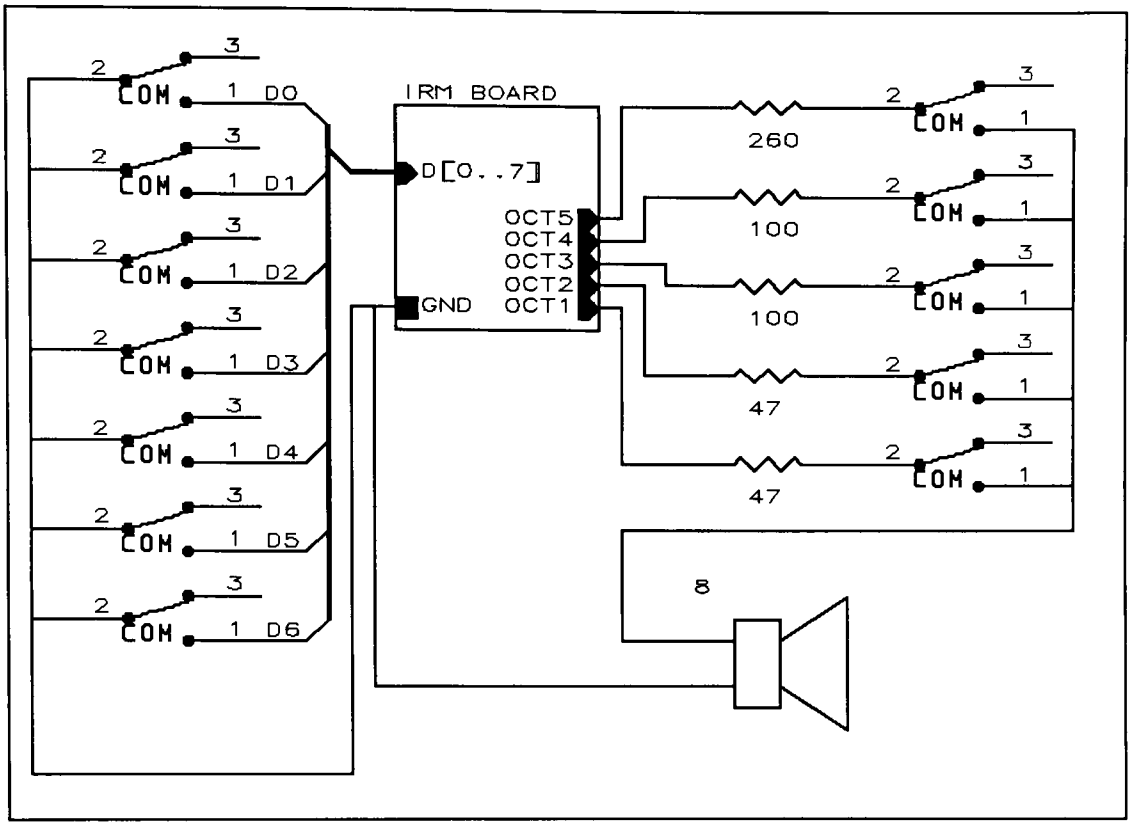

Figure 14: Prototype schematic, top sheet.

Although the circuit board can handle 8 bits of input, only seven input switches are attached. (I only needed seven switches to get the diatonic scale.) The output appears on five different octaves simultaneously. I weighted each octave with a different resistor value (to boost the audibility of the lower octaves) and brought each octave to a different switch. Any octave or combination of octaves may be selected.

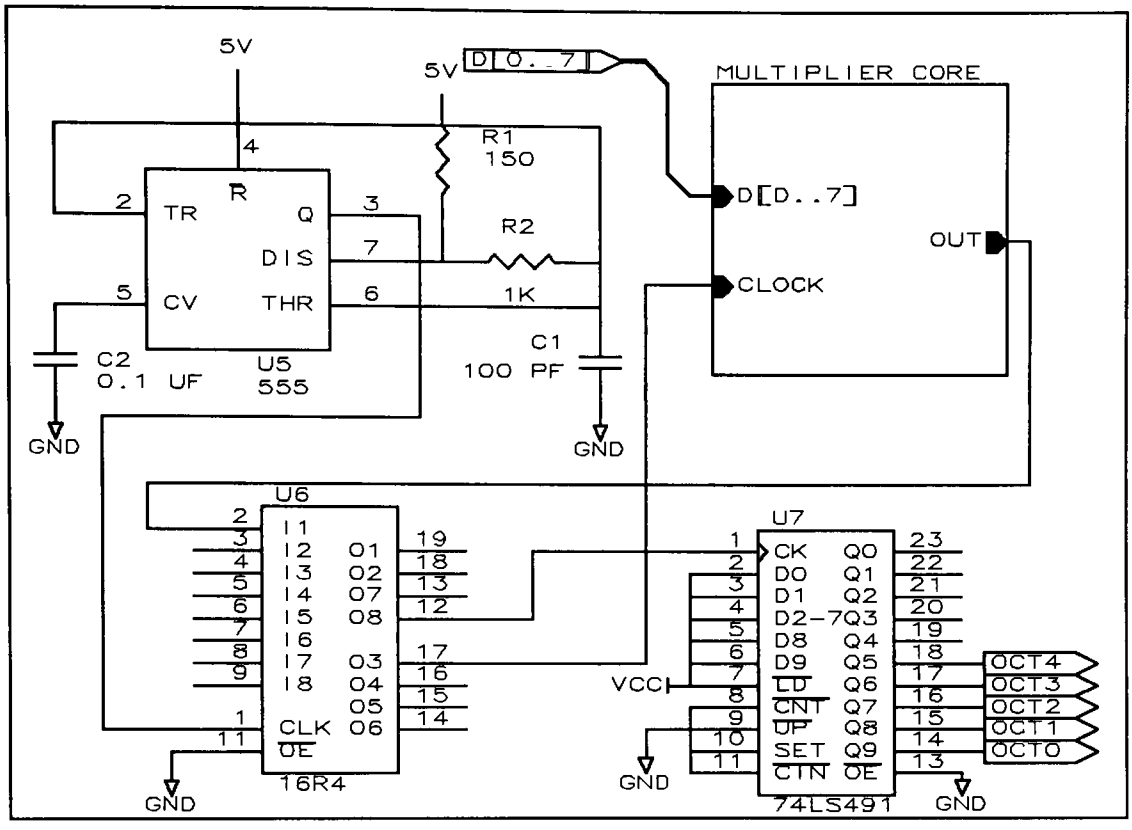

Figure 15: Prototype schematic, IRM board.

The circuit board consists of an oscillator (555), some glue logic (16R4) to generate two phases of clock and mix output pulses with the main clock (to modulate all frequencies into the same octave,) the multiplier core, and a ten bit binary counter to smooth jitter and output several octaves.

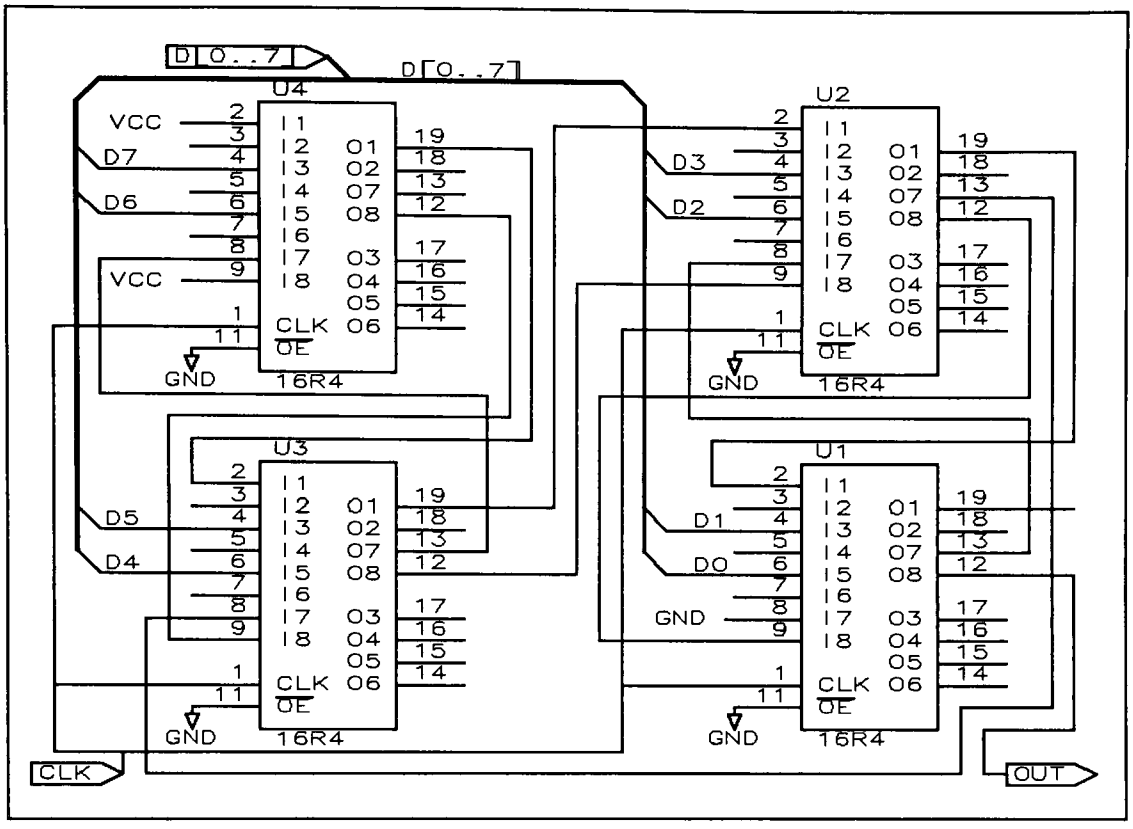

Figure 16: Prototype schematic, multiplier core.

The multiplier core consists of four 16R4 pals, chained together. At the "bottom", a constant enable is fed in to the lowest stage. At the "top", a constant "is not a normal bit" is fed in to the top stage.

1.4.2 DISTRIBUTION OF OUTPUT FREQUENCIES

One problem with this method of rate multiplication is that the 2^n frequencies produced are not evenly spaced. The reason for this is that this implementation does not handle the trivial cases efficiently. Producing a single 1 and N 0's requires N stages. The trivial case is most suited to

the use of a presettable down counter, requiring only log2(N) stages.

As n gets large, this gets very severe. In order to generate all twelve halftones, only 15 stages would be required. However, to get 1-cent resolution throughout the whole octave, on the order of 1000 stages would be required.

Is this a severe problem? It is annoying, but there seems to be no obvious way around it without sacrificing the uniqueness of the input codes. Enumerating the rationals in a way that distributes the values more equitably is a tricky problem, and I have not found a solution that lends itself to physical implementation.

2. APPENDICES

2.1 APPENDIX 1: CURIOSITIES

My ideal rate multiplier can generate approximations to some irrational numbers. One of them is "heavenly" and the other one is "devilish."

2.1.1 A HEAVENLY RATIO

The pattern generated by a code word of all ones has a curious "heavenly" property. Developing this code word proceeds like this:

Table A1-1: Ideal rate multiplier output with code word set to all ones.

input code	output pattern	1:0 ratio	output freq	octave bound
1	10	1:1	1:2	3:2
11	011	2:1	2:3	5:3
111	10101	3:2	3:5	8:5
1111	01101101	5:3	5:8	13:8
11111	1010110101101	8:5	8:13	21:13
111111	01101101011011010110	13:8	13:21	34:21

The third ratio is the second ratio plus 1. The reciprocal of the values in the third ratio appear in the next row as the second ratio. However, as code length goes to infinity, both values stabilize. So, letting x be equal to the third ratio, this ratio satisfies

$$1/x + 1 = x, \text{ or}$$

$$1 + x = x^2.$$

The positive solution of this equation yields the so-called "golden ratio."

The sequence {1,1,2,3,5,8,13,...} is the Fibonacci series, related to the golden ratio, an also to exponential growth.

2.1.2 A DEVILISH RATIO

A code word of alternating ones and zeros is also devilishly interesting. Here is a development of this ratio:

Table A1-2: Ideal rate multiplier output for codewords that start with one or two 1's, then alternate 0's and 1's and end in a 0.

input code	output pattern	1:0 ratio	output freq	octave bound
1	10	1:1	1:2	3:2
10	100	1:2	1:3	4:3
110	01010	2:3	2:5	7:5
1010	0101010	3:4	3:7	10:7
11010	100101001010	5:7	5:12	17:12
101010	10010100101001010	7:10	7:17	24:17
1101010	---	12:17	12:29	41:29
10101010	---	17:24	17:41	58:41
110101010	---	29:41	29:70	99:70
1010101010	---	41:58	41:99	140:99

Now, a few observations. The products of consecutive pairs in the octave column are equal to two.

$$3/2 * 4/3 = 2,$$

$$7/5 * 10/7 = 2,$$

$$17/12 * 24/17 = 2, \text{ and so on.}$$

Also, the values converge as the series moves out. As the sequence goes towards infinity, the octave bounded ratio converges to the square root of two. Musically, the square root of two is known as the "tri-tone" or "devil's tone". The ideal rate multiplier was designed to produce just tempered intervals, but the "devil's tone" is an equal tempered interval.

It is an added curiosity that this circuit is able to easily produce digital approximations to both the "most heavenly" ratio and the "most devilish" ratio. I have not explored this property of the ideal rate multiplier in depth, but I believe that there are more irrational numbers waiting to be found.

2.2 APPENDIX 2: THE BINARY RATE MULTIPLIER

An understanding of gray code helps in understanding the BRM. Figure 17 shows gray code compared with straight binary. Note that in the gray code counter, only one bit makes a transition on each increment.

Also, note that there is only one positive-going transition per increment for the straight binary counter, and that this transition happens on the same bit that is making a transition in the gray code counter.

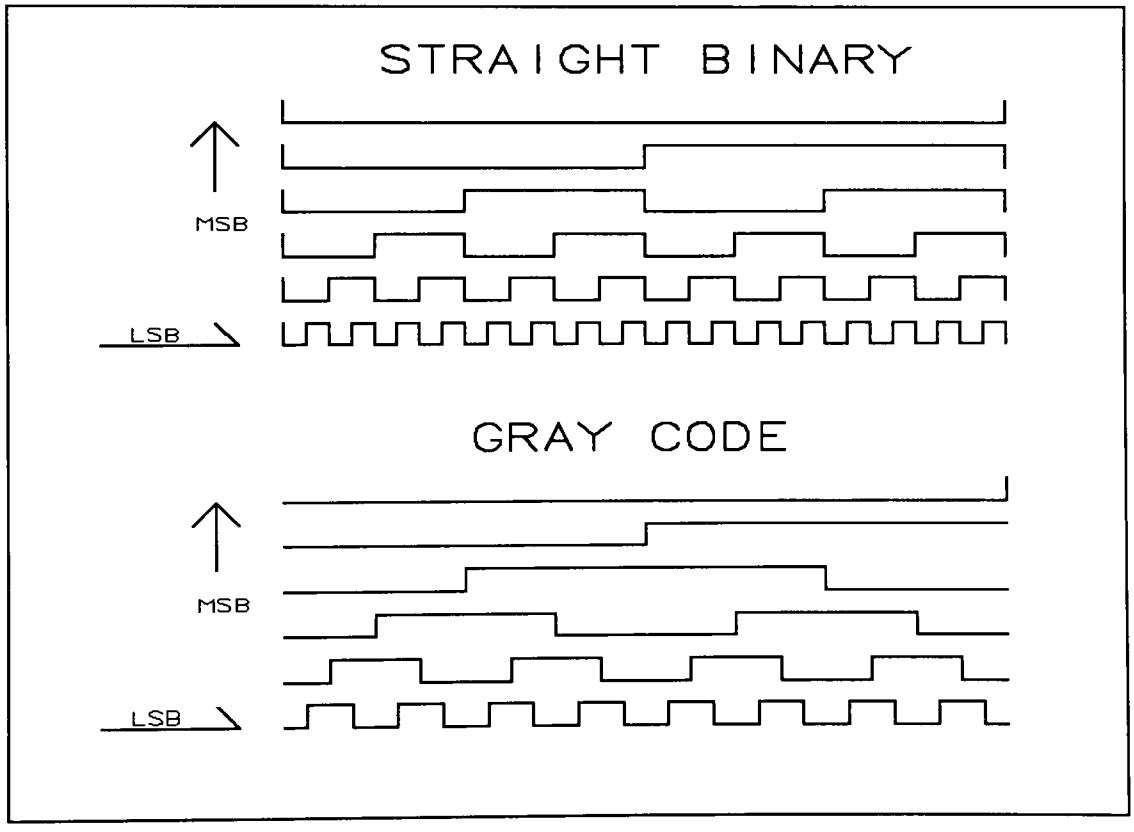

Figure 17: Straight binary versus gray code.

The least significant bit in the gray code counter makes a transition every other increment. The next least significant bit makes transitions on half of the remaining increments, and so on. Each transition on a more significant bit is exactly one increment away from a transition on bit 0 (the least significant bit), two increments away from a transition on bit 1, four increments away from a transition on bit 2, and so on. In general, any transition on bit m is exactly 2^n increments away from a transition on bit n if m > n.

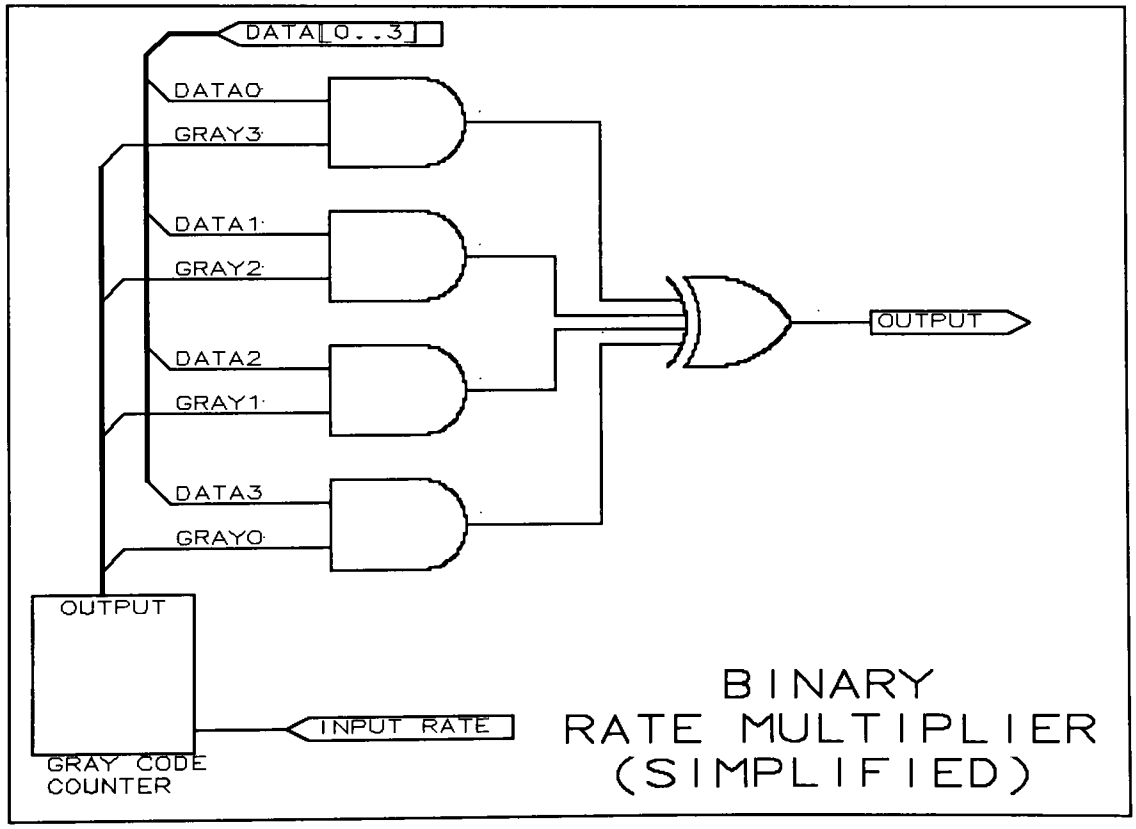

Figure 18: Simple four bit binary rate multiplier

By selectively exclusive-oring bits from a gray code counter, (see figure 18) an output can be produced that contains any number of transitions (up to $2^n - 1$) per cycle (2^n) of the counter (n-bit). Although the transitions are not as evenly spaced as possible, they are best case considering that here, each data bit turns transitions on or off in fixed positions. Transition-wise, the counter's least significant bit is most significant. (It has the most transitions on it.) The most significant data bit gates the least significant counter bit into an n input exclusive-or gate, and the other data bits are connected to counter bits in a similar manner. One could also consider this circuit to be a Walsh function generator.

The 7497 does not operate internally in this manner. Transitions on the data inputs can cause transitions on the output in this simplified gray code version. This would be a serious problem in almost any computational circuit. 7497 circuits avoid extra transitions by keeping the output low when the clock is low and clocking the counter on the trailing edge of the clock. This guarantees the output is stable while the rest of the circuit settles.

2.2.1 7497 TTL BINARY RATE MULTIPLIER

The 7497 TTL IC is a BRM. It has six digital control inputs and a top rate clock input. The top rate clock drives a six bit binary counter through all of its sixty-four

states. When any counter stage is about to make a positive going transition, it signals this state. If the data bit corresponding to this stage is set, a pulse is enabled at the output. The counter is clocked on the trailing edge of the pulses, so that glitches don't appear at the output.

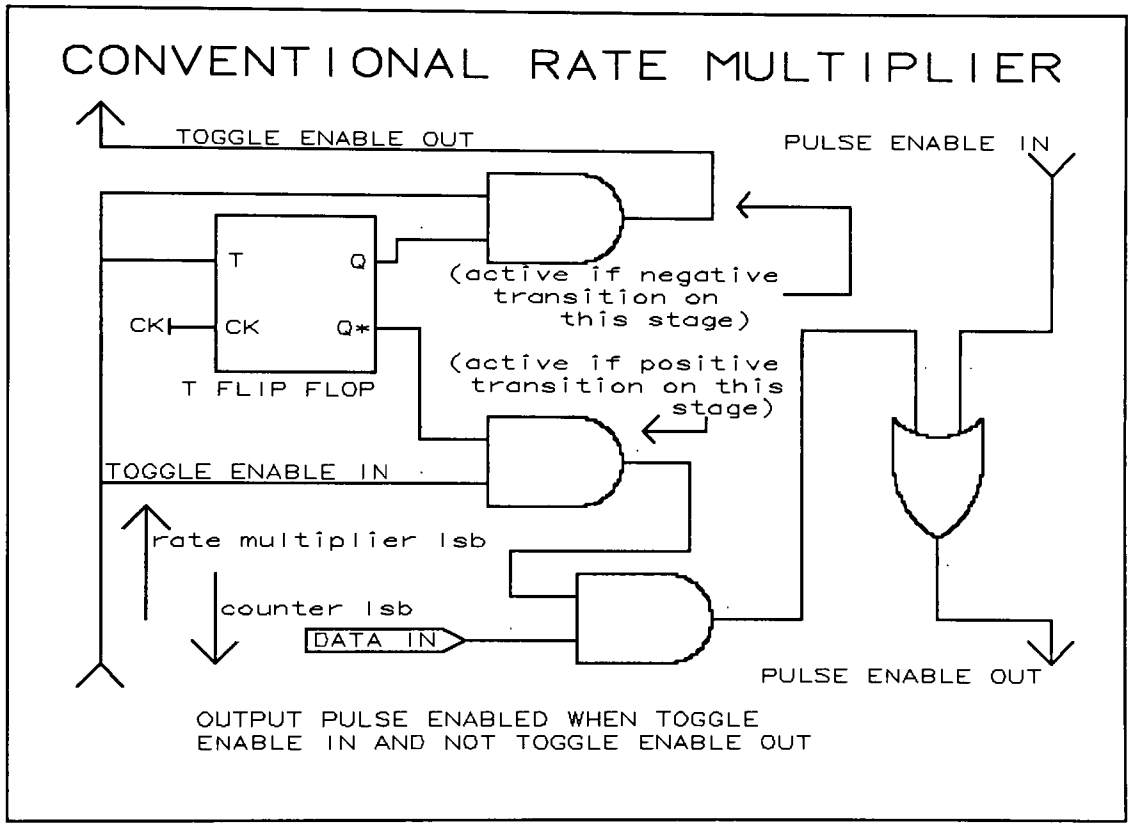

Figure 19: Single bit BRM (simplified.)

The decision to put six bits of rate multiplier in one package was cosmetic. If the binary rate multiplication algorithm used in the 7497 were put in a one bit package,[8] it would look like figure 18. Chaining six of them together,

changing ripple logic to look-ahead logic (for speed), and adding support circuitry would result in a 7497 equivalent.

2.2.2 AN EXAMPLE OF 7497 OUTPUT

If the 7497 BRM chip were instructed to output thirteen pulses, it would receive an input code of 001101. Each bit will add more pulses to the output. Here are the 6 different bit patterns:

bit 0: 00000000 00000000 00000000 00000001 (32 0's)

bit 1: 00000000 00000001 00000000 00000000 (repeats)

bit 2: 00000001 00000000 00000001 00000000 (repeats)

bit 3: 00010000 00010000 00010000 00010000 (repeats)

bit 4: 01000100 01000100 01000100 01000100 (repeats)

bit 5: 10101010 10101010 10101010 10101010 (repeats)

The output will be the or of bit 0, bit 2, and bit 3:

00010001 00010000 00010001 00010001

00010001 00010000 00010001 00010000

2.3 APPENDIX 3: PAL EQUATIONS

The following text is the pal file that generates the two stage ideal rate multiplier used in the prototype.

```
TITLE Ideal rate multiplier block. VERIFIED.
PATTERN
REVISION A
AUTHOR ANDREW MOORE
COMPANY
DATE OCTOBER 4TH 1988
; this implementation of an ideal rate multiplier models two stages.
; Each stage inputs one data bit and has
; three lines passing through it.
; The lines are: the top bit, the enable and the output.
; "Top bit" means "the first stage with a set data bit."
; Towards the top is denoted by a decreasing index.
; Towards the output = towards the bottom.
; This is denoted by an increasing index.
; Top bit enable data represents 'is not/might be'.
; A stream of top bit enable data starts out at the very top
; in the "might be" state and ripples towards the output.
; The first set data bit clears the might be.
; This stage is the 'top bit'; obviously the only one...
; So it passes an 'is not' to the next counter bit,
```

```
; which ripples uncontested to the output.
; an enable gives a stage permission
; to change its state (on the clock)
; The enable starts at the bottom
; true and ripples toward the top.
; The first stage that has its state bit set
; clears the enable.
; The output is the state bit exclusive-nored
; with the data bit.
;BASIC OPERATION:
; unset enabled stages will set.
; if a stage gets set
; upon setting, it blocks the enable line to higher stages.
;-----------------------------------------------------
CHIP NATIONAL PAL16R4

CLOCK /S0_NOT_TOP NC_THREE /S0_D NC_FIVE
/S1_D SEVEN /S1_ENABLE /SMINUS_OUT GND

/OE /S1_OUT /SMINUS_ENABLE /NC_FOURTEEN /Q1
/Q /NC_SEVENTEEN /S0_OUT /S2_NOT_TOP VCC

EQUATIONS
```

```
Q :=    ;
        ; IF IT'S A NORMAL COUNTER BIT:
        ; set bit from un-set if stage enabled and with 1
from above.
        ; a stage is enabled if the stage below is clear and
enabled.
        ; S0_ENABLE = S1_ENABLE * /Q1
        S0_NOT_TOP * /Q * S1_ENABLE * /Q1 * SMINUS_OUT
        ; keep set if not enabled
        ; /S0_ENABLE = Q1 + /S1_ENABLE (de Morgan's)
       + S0_NOT_TOP * Q * /S1_ENA
       + S0_NOT_TOP * Q * Q1
        ;this means a set bit is cleared on the next enable
        ;
        ; IF IT'S THE TOP BIT:
        ; set if enabled and unset
       + /S0_NOT_TOP * S0_D * /Q * S1_ENABLE * /Q1
        ; keep set if not enabled
       + /S0_NOT_TOP * S0_D * Q * /S1_ENA
       + /S0_NOT_TOP * S0_D * Q * Q1
       ; and, of course, clear if set and enabled:
Q1 := ;
        ; IF IT'S A NORMAL COUNTER BIT:
        ; set bit from un-set if stage enabled and with 1
from above.
```

```
       ; S1 is the top bit if the bit above might be the
top, but
            S0_NOT_TOP * /Q1 * S1_ENABLE * S0_OUT
          + S0_D         * /Q1 * S1_ENABLE * S0_OUT
           ;keep set if not enabled
          + S0_NOT_TOP * Q1 * /S1_ENA
          + S0_D         * Q1 * /S1_ENA
          ;
          ; top bit, toggle only:
          ;
          ; toggle from unset if enabled
          + /S0_NOT_TOP * /S0_D * S1_D * /Q1 * S1_ENA
          ; keep set if set and not enabled
          + /S0_NOT_TOP * /S0_D * S1_D * Q1 * /S1_ENA
          ; and set to zero otherwise.

S0_OUT = S0_D * /Q + Q * /S0_D

S1_OUT = S1_D * /Q1 + Q1 * /S1_D

SMINUS_ENABLE = S1_ENABLE * /Q1 * /Q

S2_NOT_TOP = S0_NOT_TOP + S0_D + S1_D
;
```

3. REFERENCES

[1] Burr-Brown Corporation, Burr-Brown Integrated Circuits Data Book Volume 33, pp 5-39, 5-40. 1989

[2] Campbell, R.A., personal correspondence.

[3] Gorski-Popiel, ed., Frequency Synthesis: Techniques and Applications, IEEE Press, 1975

[4] Kinter, P., Electronic Digital Techniques, McGraw-Hill, pp 258 - 271, 1968

[5] Lancaster, D., TTL Cookbook, Howard W Sams & Co, pp 284 - 291, 1974

[6] Manassewitsch, V., Frequency Synthesizers: Theory and Design, Wiley Interscience, 1987

[7] Mathews, M. V. and Pierce, J., "The Bohlen-Pierce Scale", from Current Directions in Computer Music Research, MIT press, pp. 165 - 173, 1989

[8] Norris, B., Electronic Power control and digital techniques, McGraw-Hill, pp 167-182, 1967

[9] Roberts, L. A., and Mathews, M. V., "Intonation Sensitivity for Traditional and Non-Traditional Chords", from The Journal of the Acoustical Society of America, Vol. 75, no. 3, pp 952-959, Bell Laboratories, 1984

[10] Smith, R., Harmonics or The philosophy of Musical Sounds, Da Capo Press, 1966. (originally published in Cambridge, England in 1749)

[11] Texas Instruments, ALS/AS Logic Data Book, Texas Instruments Inc., p. 1-28, 1986

[12] Texas Instruments, High-speed CMOS Logic Data Book, Texas Instruments Inc., pp. 1-6 to 1-13, 1984